Deadlock Detection and Recovery

Richard M. Fujimoto
Professor

Computational Science and Engineering Division
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0765, USA

http://www.cc.gatech.edu/~fujimoto/

Outline

- Deadlock Detection and Recovery Algorithm (Chandy and Misra)
  - Basic Approach
  - Deadlock Detection
    - Diffusing distributed computations
    - Dijkstra/Scholten algorithm (signaling protocol)
  - Deadlock Recovery

Deadlock Detection & Recovery

Algorithm A (executed by each LP):

Goal: Ensure events are processed in time stamp order:

WHILE (simulation is not over)

    wait until each FIFO contains at least one message

    remove smallest time stamped event from its FIFO

    process that event

END-LOOP

- No null messages
- Allow simulation to execute until deadlock occurs
- Provide a mechanism to detect deadlock
- Provide a mechanism to recover from deadlocks

Deadlock Detection

Diffusing computations (Dijkstra/Scholten)

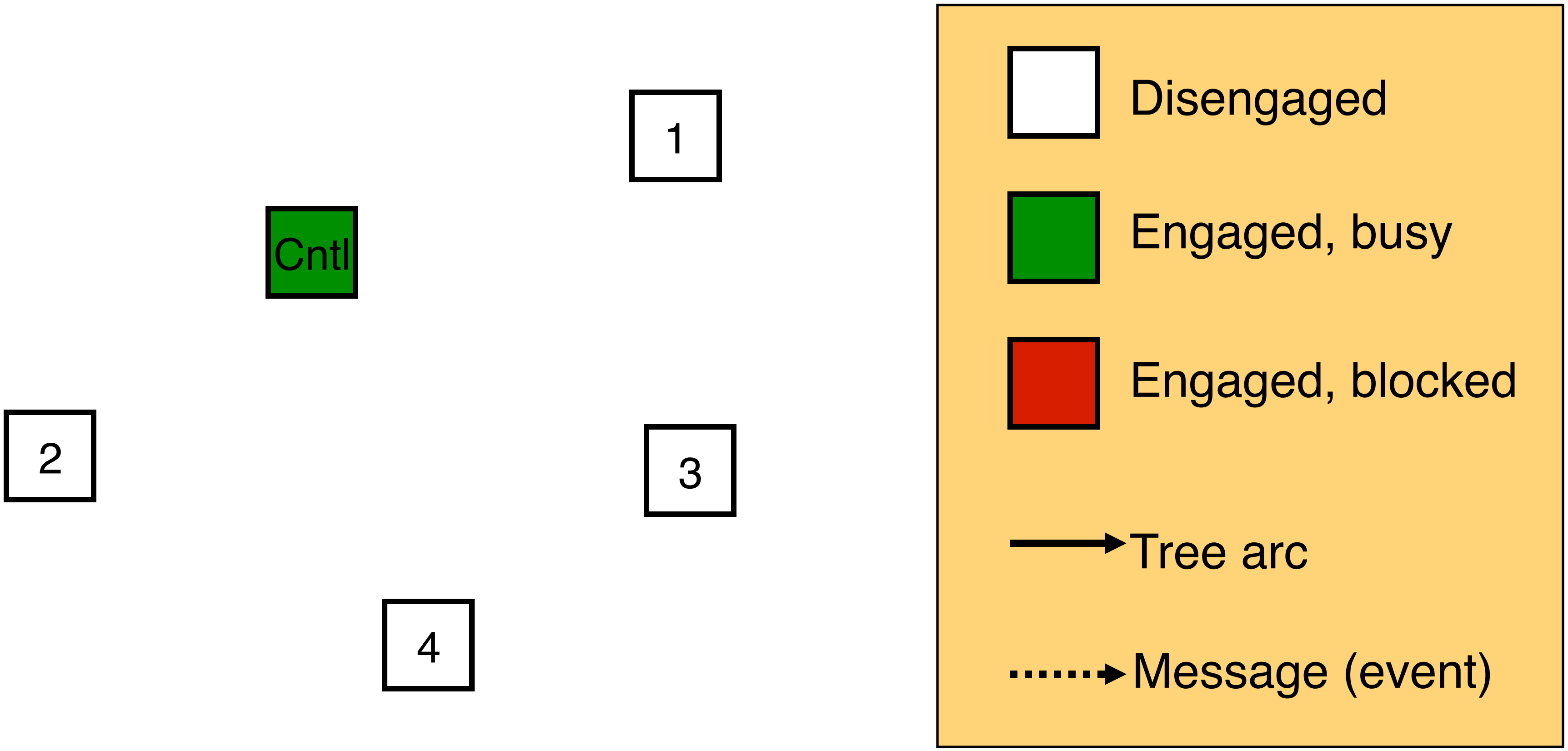
- Computation consists of a collection of processes that communicate by exchanging messages
- Receiving a message triggers computation; may result in sending/receiving more messages
- Processes do not spontaneously start new computations (must first receive a message)
- One process identified as the “controller” that is used for deadlock detection and recovery

Goal: determine when all of the processes are blocked (global deadlock)

Basic Idea

- Initially, all processes blocked except controller
- Controller sends messages to one or more processes to break deadlock
- Computation spreads as processes send messages
- Construct a tree of processes that expands as the computation spreads, contracts as processes become idle
  - Processes in tree are said to be engaged
  - Processes not in tree are said to be disengaged
- A disengaged process becomes engaged (added to tree) when it *receives a message*
- An engage process becomes disengaged (removed from the tree) when *it is a leaf node* and *it is idle (blocked)*
- If the tree only includes the controller, the processes are deadlocked

Example



3 becomes disengaged, controller begins recovery

Processes

- Engaged process
  - Has received at least one message for which no signal has been returned
  - Engaged process (***non-leaf***): process has sent one or more messages for which a signal has not yet been returned
  - Engaged process (***leaf***): A signal has been received for every message sent by the process
- Disengaged process
  - A signal has been returned for all received messages
  - A signal has been received for all messages the process has sent

Implementation: Signaling Protocol

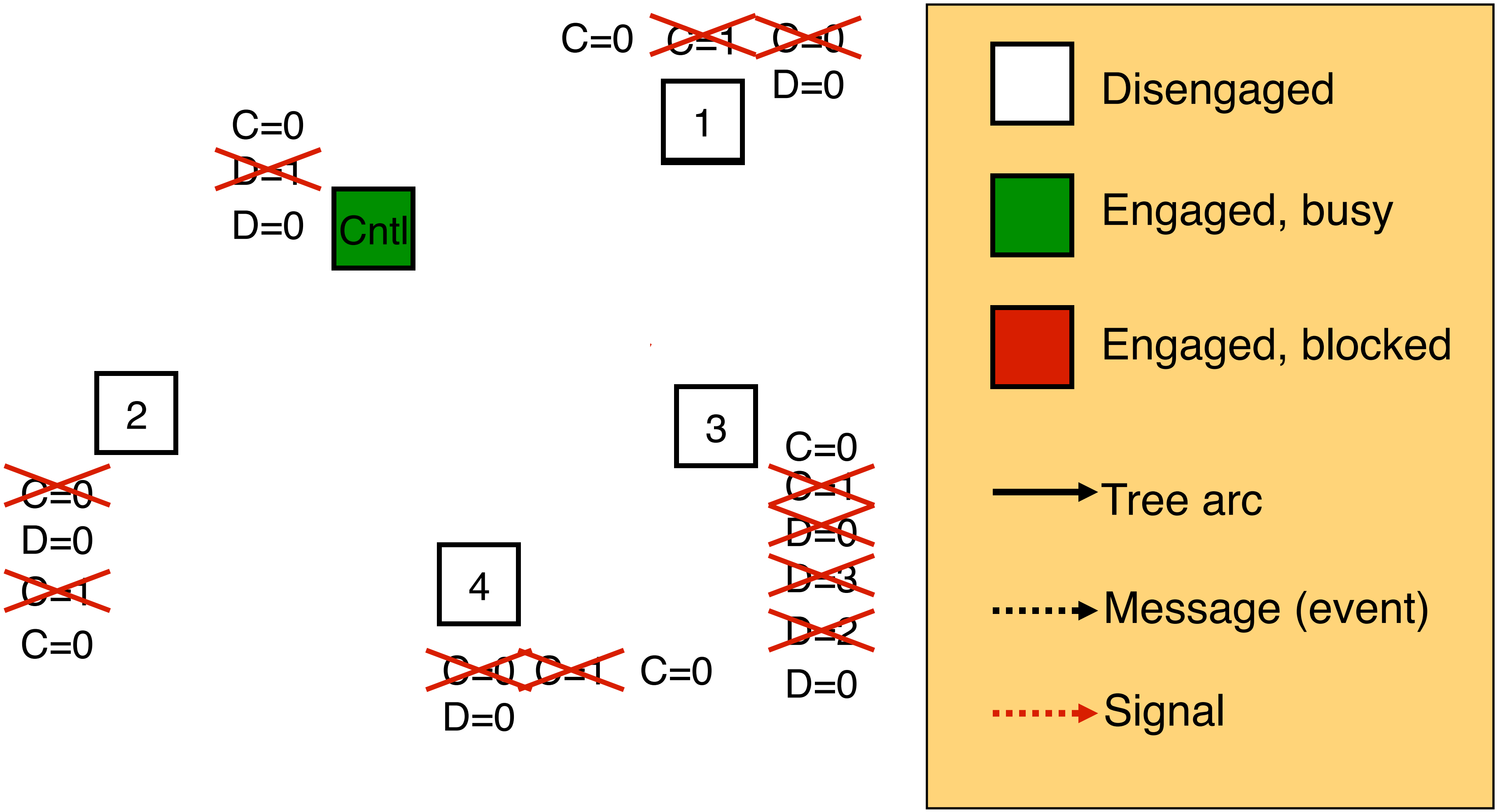
Add signaling protocol to simulation execution

- When an engaged process receives a message
  - Immediately return a signal to sender indicating message did not spawn a new node in the tree
- When a disengaged process receives a message:
  - Receiving process becomes engaged
  - Do *not* return a signal until it becomes disengaged
- An engaged process becomes disengaged (and sends a signal to its parent in the tree) when
  - It is idle, and
  - It is a leaf node in the tree
    - Process is a leaf it it has received signals for all messages it has sent

Implementation (cont.)

- Each process maintains two variables
  - C = # messages received for which process hasn't returned a signal
  - D = # messages sent for which a signal has not yet been received
- When are C and D updated?
  - Send a message: Increment D in sender, increment C in receiver
  - Return a signal: Decrement C in sender, decrement D in receiver
- When is a process disengaged?
  - A process is disengaged if C = D = 0
- When is a process a leaf node of tree?
  - A process is a leaf node if C>0, D=0
- When does a process send a signal?
  - If C=1, D=0, and the process is idle (becomes disengaged), or
  - If it receives a message and C>0
- When is deadlock detected?
  - System deadlocked if C = D = 0 in the controller

Example



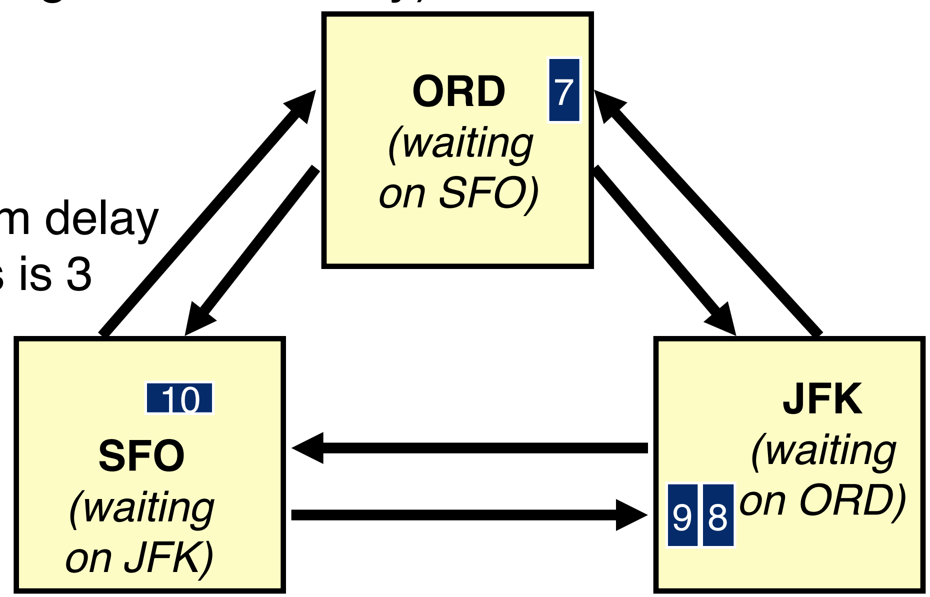
controller detects deadlock, begins recovery

Deadlock Recovery

Deadlock recovery: identify “safe” events (events that can be processed w/o violating local causality),

deadlock state

Assume minimum delay between airports is 3



Which events are safe?

- Time stamp 7: smallest time stamped event in system
- Time stamp 8, 9: safe because of lookahead constraint
- Time stamp 10: OK if events with the same time stamp can be processed in any order
- No time creep!

Summary

- Deadlock Detection
  - Diffusing computation: Dijkstra/Scholten algorithm
  - Simple signaling protocol detects deadlock
  - Does not detect partial (local) deadlocks
- Deadlock Recovery
  - Smallest time stamp event safe to process
  - Others may also be safe (requires additional work to determine this)